Reinforcement Learning for Edit-Based Non-Autoregressive Neural Machine Translation

Hao Wang[1*] **Tetsuro Morimura**[2] **Ukyo Honda**[2] **Daisuke Kawahara**[1]
[1] Waseda University [2] CyberAgent
[1] {conan1024hao@akane., dkw@}waseda.jp
[2] {morimura_tetsuro, honda_ukyo}@cyberagent.co.jp

Abstract

Non-autoregressive (NAR) language models are known for their low latency in neural machine translation (NMT). However, a performance gap exists between NAR and autoregressive models due to the large decoding space and difficulty in capturing dependency between target words accurately. Compounding this, preparing appropriate training data for NAR models is a non-trivial task, often exacerbating exposure bias. To address these challenges, we apply reinforcement learning (RL) to Levenshtein Transformer, a representative edit-based NAR model, demonstrating that RL with self-generated data can enhance the performance of edit-based NAR models. We explore two RL approaches: stepwise reward maximization and episodic reward maximization. We discuss the respective pros and cons of these two approaches and empirically verify them. Moreover, we experimentally investigate the impact of temperature setting on performance, confirming the importance of proper temperature setting for NAR models' training.

1 Introduction

Non-autoregressive (NAR) language models (Gu et al., 2018) generate translations in parallel, enabling faster inference and having the potential for real-time translation applications. However, despite their computational efficiency, NAR models have been observed to underperform autoregressive (AR) models due to the challenges posed by the large decoding space and difficulty in capturing dependency between target words accurately (Gu et al., 2018). To bridge the performance gap, many NAR architectures and training methods have been proposed, including edit-based models like Insertion Transformer (Stern et al., 2019) and Levenshtein Transformer (Gu et al., 2019). Prior research has also explored knowledge distillation (Ghazvininejad et al., 2019), which is effective but introduces additional complexity.

Unlike AR models, preparing teacher data and designing appropriate training objectives have always been challenging for NAR models (Li et al., 2023). Teacher forcing with inappropriate teacher data may exacerbate the exposure bias problem (Ranzato et al., 2016), affecting model performance. Reinforcement learning (RL) is known for its ability to tackle the exposure bias (Ranzato et al., 2016) and alleviate the object mismatch issue (Ding and Soricut, 2017). Despite its importance, explorations of RL for NAR are still scarce. Shao et al. (2021) proposed a method for reducing the estimation variance. However, this method is only applicable to NAR models with a fixed output length, which is unsuitable for edit-based models.

In this paper, we empirically analyze conditions for performance improvement in applying RL to edit-based NAR models in neural machine translation (NMT). Specifically, we focus on Levenshtein Transformer (LevT) (Gu et al., 2019), a prominent edit-based NAR architecture that has shown promise in reducing decoding latency and flexible length adjustment. We demonstrate that RL with self-generated data significantly improves LevT's performance. Importantly, our methods are orthogonal to existing research on NAR architectures, indicating potential for widespread applicability. We explore two RL approaches: stepwise reward maximization, which computes rewards after each edit operation, and episodic reward maximization, which only computes rewards after all generations are completed. We analyze these two approaches' respective advantages and disadvantages and empirically verify them. Furthermore, through a series of experiments, we investigate the impact of temperature settings on softmax sampling, aiming to identify the optimal temperature that strikes a balance between exploration and exploitation during the RL training process.

[*] Work done during internship at CyberAgent AI Lab.

Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 4: Student Research Workshop), pages 212–218

2 Background

Reinforcement Learning Reinforcement learning has been widely applied to improve the performance of AR NMT models (Ranzato et al., 2016; Bahdanau et al., 2016; Wu et al., 2016) because its ability to train models to optimize non-differentiable score functions and tackle the exposure bias problem (Ranzato et al., 2016). In practice, REINFORCE (Williams, 1992) with a baseline is commonly used for estimating the policy gradient, which can be computed as follows:

$$\bigtriangledown_\theta L(\theta) \approx -(r(y) - b(s)) \bigtriangledown_\theta \log \pi_\theta(y|s), \quad (1)$$

where r is the reward function, b is the baseline, y is a sample from policy π_θ and state s.

Softmax with Temperature In the domain of RL, we need to consider the exploration-exploitation trade-off (Sutton and Barto, 2018), where temperature τ is an important parameter. τ is used to control the softness of the softmax distribution,

$$p_i = \frac{\exp(y_i/\tau)}{\sum_i \exp(y_i/\tau)}. \quad (2)$$

A larger τ leads to a more uniform distribution, promoting exploration, while a smaller τ creates a more peaky distribution, emphasizing exploitation.

Kiegeland and Kreutzer (2021) shows that training with an increased temperature can mitigate the peakiness effect due to RL (Choshen et al., 2020), indicating that a suitable temperature is significant for RL training in NMT.

RL for NAR Compared to AR methods, studies of reinforcement learning for NAR remain unexplored. Shao et al. (2021) proposed a method to reduce the estimation variance of REINFORCE by fixing the predicted word at position t and sampling words of other positions for n times. However, this method is only applicable to models with a fixed length, which is unsuitable for edit-based models.

Levenshtein Transformer Levenshtein Transformer (Gu et al., 2019) is an NAR model based on three edit operations: delete tokens, insert placeholders, and replace placeholders with new tokens. It uses a supervised dual-policy learning algorithm to minimize the Levenshtein distance (Levenshtein, 1965) for training and greedy sampling for decoding. The decoding stops when two consecutive refinement iterations return the same output or a maximum number of iterations (set to 10) is reached. We illustrate the decoding process in Figure 1.

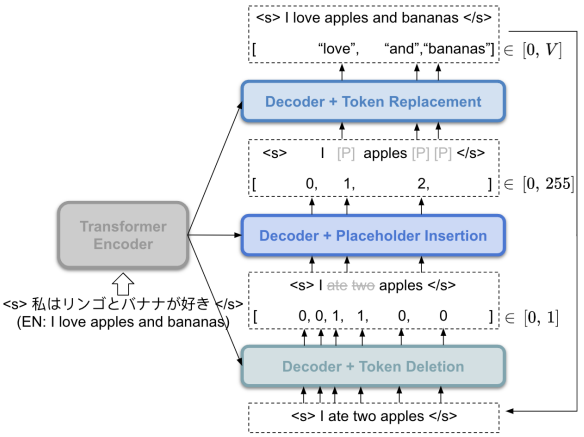

Figure 1: The illustration of Levenshtein Transformer's decoding process (Gu et al., 2019). In each decoding iteration, three edit operations are performed sequentially: delete tokens, insert placeholders, and replace placeholders with new tokens.

LevT's dual-policy learning generates teacher data by corrupting the ground truth and reconstructing it with its adversary policy. This mechanism not only offers a unique approach to data generation but also underscores the inherent difficulty in preparing teacher data. This introduces concerns regarding the exposure bias, particularly whether the training process can maintain consistency with the text during decoding. To address this issue, we employ RL approaches that use self-generated data for training.

3 Approaches

In this section, we present our reinforcement learning approaches in detail. We train a Levenshtein Transformer model as our baseline using the dual-policy learning algorithm. Based on it, we introduce two distinct RL approaches within the REINFORCE framework: stepwise reward maximization and episodic reward maximization. Moreover, we present our methods for temperature control.

Stepwise Reward Maximization General RL training methods for AR NMT models are all episodic[1], as it is difficult to calculate BLEU (Papineni et al., 2002) when the sentence is not fully generated. In contrast, NAR models can calculate BLEU on outputs at each decoding step. From the perspective of estimating a more accurate gradient, we propose stepwise reward maximization, which

[1]In this context, "episodic" denotes training based on entirely generated sequences

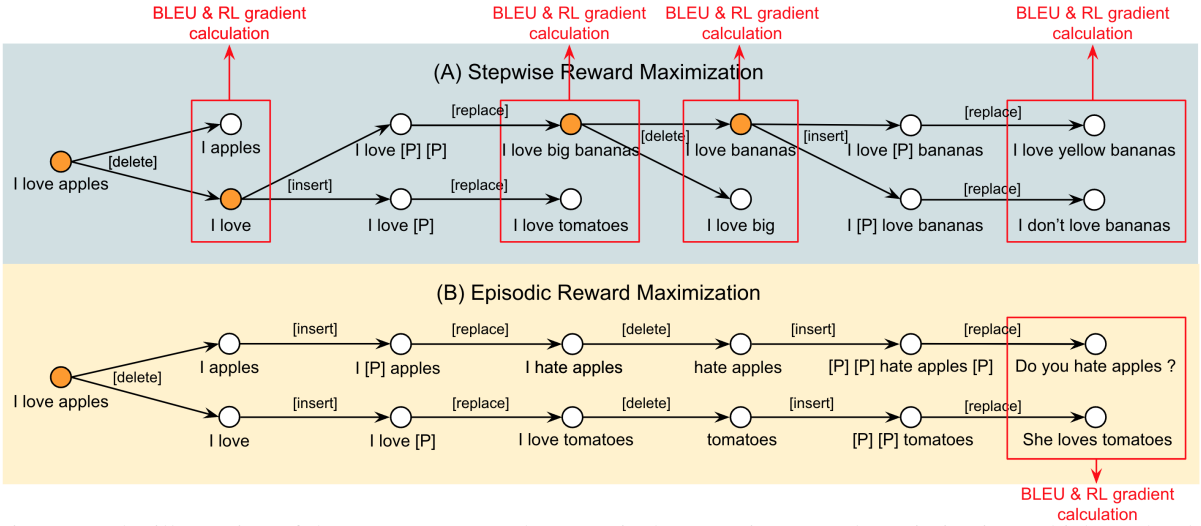

Figure 2: The illustration of the two RL approaches. (A) is the stepwise reward maximization, which randomly samples from a previous node for each edit operation and calculates BLEU and RL gradient after each edit operation (except for the insert operation, since it is not easy to calculate BLEU after inserting placeholders). (B) is the episodic reward maximization, where each sample is edited multiple times in a linear fashion, without branching into different paths, and BLEU and RL gradient are calculated only after the completion of all edit operations. At every orange node, we sample k times from this node (in this example, the sample size k is 2).

calculates reward for each edit operation[2] using score differences from one previous edit. Since every step's reward is calculated separately, this approach should be easier to learn than episodic approaches (Sutton and Barto, 2018). However, it is also more prone to learning bias since the editing process is inherently multi-step. This drawback should not be emphasized since maximizing the reward for each step will likely maximize the episodic reward in NAR models' training.

We use a leave-one-out baseline (Luo, 2020) for $b(s)$ in Equation 1 instead of the greedy baseline proposed in SCST (Rennie et al., 2017) because the greedy decoding is too strong in LevT, which makes gaining positive rewards in SCST difficult and may reduce learning efficiency. For each edit, we sample k actions from the policy at this point. Then, we calculate the baseline as follows:

$$b_i(s) = \frac{1}{k-1} \sum_{j \neq i} r(y_j), \tag{3}$$

where y_j is the jth sample from the current policy. The final RL gradient estimation becomes

$$\bigtriangledown_\theta L(\theta) \approx -(r(y_i) - b_i(s)) \bigtriangledown_\theta \log \pi_\theta(y_i|s). \tag{4}$$

In a straightforward implementation, one might consider applying sampling again to all k samples from the last edit. However, this will cause a combination explosion when the number of edit operations increases. Practically, we randomly choose a sample from the previous edit to perform the subsequent operations. We show an illustration of the sampling process in (A) of Figure 2 and pseudo code of our algorithm in Appendix A.

Episodic Reward Maximization We also introduce episodic reward maximization, which calculates rewards only once for each sample and gives all actions the same weight. It is a more traditional way to train NMT models in RL. It allows unbiased learning but may not be efficient.

We use the leave-one-out baseline for the episodic reward as well as the stepwise reward. We sample k samples from the initial input. Each sample will be edited multiple times without a branch. After the final edit, we calculate the rewards and baselines. We show an illustration of the sampling process in (B) of Figure 2 and pseudo code of our algorithm in Appendix B.

Temperature Control Applying RL to NAR differs significantly from AR because there could be various types of actions rather than just predicting the next token, like deletion and insertion. Due to this difficulty, NAR may need more fine-grained temperature control during training. To investigate the impact of exploration and exploitation in the training process, we explore five different settings of the temperature. Due to the large decoding space

[2]In practice, since it is not easy to calculate BLEU after inserting placeholders, we consider placeholder insertion and token replacement as one edit operation.

of Levenshtein Transformer, default temperature 1 may result in poor rewards, and too small temperature may result in peaky distribution, which are both harmful to learning. We use three constant temperature settings set to 0.1, 0.5, and 1 to verify the effect of temperature magnitude.

An annealing schedule is known for balancing the trade-off between model accuracy and variance during training (Jang et al., 2016). There are two ways of thinking here. First, to reduce the exposure bias, we want to get close to the decoding scenario, which is greedy decoding in our experiments. Thus, we can apply a regular annealing schedule to gradually reduce the temperature from 1 to 0.1 during training. The temperature function can be written as follows:

$$\tau_{i+1} = \max(\tau_i * \exp(-\frac{\log(\tau_0/\tau_T)}{T}), \tau_T), \quad (5)$$

where T is the number of total training steps, and τ_0 and τ_T are the initial and the target temperatures.

Second, using high temperatures in the early stages of training may lead to poor rewards and result in low learning efficiency. We can apply an inverted annealing schedule to gradually increase the temperature from 0.1 to 1, guaranteeing stable training in the early stages and gradually increasing the exploration space for efficient training. The temperature function can be written as follows:

$$\tau_{i+1} = \min(\tau_i/\exp(-\frac{\log(\tau_T/\tau_0)}{T}), \tau_T). \quad (6)$$

In each decoding iteration, multiple edit operations occur, and each operation has a different decoding space size. It may be beneficial to optimize this by using varying temperatures for each operation in every iteration. This is a complicated research question and we leave this exploration to future work.

4 Experiments

4.1 Experimental Setup

Data & Evaluation We use WMT'14 English-German (EN-DE) (Bojar et al., 2014) and WAT'17 English-Japanese (EN-JA) Small-NMT datasets (Nakazawa et al., 2017) for experiments. We use BPE token-based BLEU scores for evaluations. Data preprocessing follows Gu et al. (2019).

Baseline We use Levenshtein Transformer as our baseline. Following Gu et al. (2019), we trained a LevT with 300K steps and a max batch size of 65,536 tokens per step. However, like Reid et al. (2023), we cannot reproduce the results of Gu et al. (2019). We use our results in this paper.

RL According to Gu et al. (2019), most decodings are gotten in 1-4 iterations, and the average number of decoding iterations is 2.43. To minimize the gap between the training and decoding states, we start with a null string and conduct 3 iterations (8 edits) for each sample during RL training. We set the total training steps T to 50,000, with a max batch size of 4,096 tokens per step. To prevent the out-of-memory issue, we limit the decoding space of placeholder insertion from 256 to 64. The sample size k of the baseline is set to 5. Our implementation is based on Fairseq[3].

Computational Cost The pre-training phase of LevT on a GCP VM instance with A100x4 GPUs requires roughly 3 days, while the subsequent RL fine-tuning process takes approximately 1 day to complete.

4.2 Results

We show the BLEU scores of our approaches in Table 1. The episodic reward model[4] showed notable improvement over the baseline. The score is even close to the distillation model, which requires a heavy pre-training[5] of AR models. However, the stepwise reward model showed only limited improvement. To explain this, we focus on the advantage, $r(y) - b(s)$, included in the policy gradient (Equation 1), as a larger value of the advantage can increase the policy gradient's magnitude. A higher standard deviation (SD) of the advantages indicates larger fluctuations in policy gradients. Table 2 shows the SDs of the advantages of the stepwise reward model, with notably higher values in the early stages of edit operations compared to later stages. This suggests that the stepwise reward model disproportionately focuses on early operations, potentially leading to uneven learning and reduced performance. In contrast, the episodic reward model applies the same rewards and advantages across all operations, facilitating more uniform learning and improved performance.

[3] https://github.com/facebookresearch/fairseq

[4] The term "episode/stepwise reward model" specifically refers to the model trained using the "episode/stepwise reward maximization" approach.

[5] To produce a distillation model, we need to train an autoregressive Transformer first, which needs additional 3 days of training on our machine.

Model	EN-DE	EN-JA
LevT	24.03	31.76
LevT + distillation	26.49	-
LevT + RL (stepwise)	24.29	31.73
LevT + RL (episodic)	**25.72**	**32.75**

Table 1: The BLEU scores of our approaches and the baseline. Temperatures are set to 1. Due to the limited computational resources, we only trained the distillation model for the EN-DE dataset using the ready-made distillation dataset.

Iteration	Edit Operation	EN-DE	EN-JA
1	Insert + Replace	9.99	8.59
2	Delete	2.05	1.35
	Insert + Replace	3.28	2.48
3	Delete	1.67	1.29
	Insert + Replace	3.04	1.60

Table 2: Stepwise reward model's standard deviation (SD) of the advantage in each edit operation. Insertion and replacement share the same reward.

We only report scores of applying RL to the model without distillation since we found that RL significantly improved the model without distillation (max 1.69 points) compared to when distillation was applied (max 0.5 point). Moreover, when confronted with distillation models, it raises questions such as which data we should use for RL training, the original or the distillation one. We leave these research questions to future work.

We show the BLEU scores of different temperature settings in Table 3. Model performance varies significantly with temperature settings (max 1.01 points in EN-JA). Among constant setting models, the model with a temperature of 0.5 performed best in EN-DE, and the model with a temperature of 0.1 performed best in EN-JA, indicating that too large temperature harms RL training. The two models using annealing schedules performed great in both tasks, showing the effectiveness of the annealing algorithms for improving learning efficiency. However, the annealing models did not always outperform the constant models, which suggests the difficulty of seeking the optimal temperature setting for NAR models' RL training. Also, we found the inverted annealing model (τ=0.1→1) begins dropping performance after 10,000 steps training in EN-JA, indicating that the speed of annealing will significantly affect the model training quality.

Temperature	EN-DE	EN-JA
Constant ($\tau = 1$)	25.72	32.75
Constant ($\tau = 0.5$)	**25.98**	33.45
Constant ($\tau = 0.1$)	25.76	33.60
Annealing ($\tau = 1 \rightarrow 0.1$)	25.83	**33.76**
Annealing ($\tau = 0.1 \rightarrow 1$)	25.90	33.43

Table 3: The BLEU scores of episodic reward models using different temperature settings.

We also quickly surveyed the relationship between performance and the number of decoding iterations in RL. The model performance dropped when we reduced the number of iterations to 2 during training and remained flat when we increased it to 4, indicating that our setting is reasonable.

5 Conclusion and Future Work

This paper explored the application of reinforcement learning to edit-based non-autoregressive neural machine translation. By incorporating RL into the training process, we achieved a significant performance improvement. By empirically comparing stepwise and episodic reward maximization, we analyzed the advantages and disadvantages of these RL approaches. We plan to have a deeper exploration of stepwise reward maximization and find a way to alleviate training inequality for multiple edit operations in the future.

Our investigation of temperature settings in NAR softmax sampling provided insights into striking a balance between exploration and exploitation during training. Although our annealing methods perform well, they are not optimal and still depend on manually adjusting the parameters such as total training steps. In the future, we plan to develop a self-adaption temperature control method using various indicators like entropy and advantage SD.

The experiments in this paper focused on the basics, and we plan to do more study for practical applications in future work. As our methods are orthogonal to existing research on NAR architectures, our next step involves exploring the methods' applicability across a broader spectrum, including state-of-the-art models. Additionally, we plan to investigate how to effectively apply RL to the distillation model, the impact of different baseline designs on performance, and the impact of RL on output diversity. Applying RL to NAR is a massive and complex research question. We look forward to more researchers joining this topic.

References

Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. 2016. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086*.

Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Aleš Tamchyna. 2014. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, Maryland, USA. Association for Computational Linguistics.

Leshem Choshen, Lior Fox, Zohar Aizenbud, and Omri Abend. 2020. On the weaknesses of reinforcement learning for neural machine translation. In *International Conference on Learning Representations*.

Nan Ding and Radu Soricut. 2017. Cold-start reinforcement learning with softmax policy gradient.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. 2019. Mask-predict: Parallel decoding of conditional masked language models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6112–6121, Hong Kong, China. Association for Computational Linguistics.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. 2018. Non-autoregressive neural machine translation.

Jiatao Gu, Changhan Wang, and Junbo Zhao. 2019. Levenshtein transformer. *Advances in Neural Information Processing Systems*, 32.

Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Samuel Kiegeland and Julia Kreutzer. 2021. Revisiting the weaknesses of reinforcement learning for neural machine translation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1673–1681, Online. Association for Computational Linguistics.

Vladimir I. Levenshtein. 1965. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics. Doklady*, 10:707–710.

Yifan Li, Kun Zhou, Wayne Xin Zhao, and Ji-Rong Wen. 2023. Diffusion models for non-autoregressive text generation: A survey.

Ruotian Luo. 2020. A better variant of self-critical sequence training.

Toshiaki Nakazawa, Shohei Higashiyama, Chenchen Ding, Hideya Mino, Isao Goto, Hideto Kazawa, Yusuke Oda, Graham Neubig, and Sadao Kurohashi. 2017. Overview of the 4th workshop on Asian translation. In *Proceedings of the 4th Workshop on Asian Translation (WAT2017)*, pages 1–54, Taipei, Taiwan. Asian Federation of Natural Language Processing.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2016. Sequence level training with recurrent neural networks.

Machel Reid, Vincent Josua Hellendoorn, and Graham Neubig. 2023. DiffusER: Diffusion via edit-based reconstruction. In *The Eleventh International Conference on Learning Representations*.

Steven J. Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. 2017. Self-critical sequence training for image captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Chenze Shao, Yang Feng, Jinchao Zhang, Fandong Meng, and Jie Zhou. 2021. Sequence-level training for non-autoregressive neural machine translation. *Computational Linguistics*, 47(4):891–925.

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. 2019. Insertion transformer: Flexible sequence generation via insertion operations. In *International Conference on Machine Learning*, pages 5976–5985. PMLR.

Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*, second edition. The MIT Press.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, pages 5–32.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation.

A Pseudo code of stepwise reward maximization

We show pseudo code of stepwise reward maximization in Figure 3.

```
model = LevenshteinTransformer()
sampling_size = 5
iteration = 3
src_tokens = "input tokens"
tgt_tokens = "target tokens"

encoder_out = model.encoder(src_tokens)
prev_decoder_out = model.initialize_output_tokens()

decoder_outs = []
policies = []

for step in range(iteration):
    for sample_idx in range(sampling_size):
        output_tokens, del_policy = model.delete_tokens(prev_decoder_out, encoder_out)
        decoder_outs.append(output_tokens)
        policies.append(del_policy)
    prev_decoder_out = random.choice(decoder_outs[-sampling_size:])

    for sample_idx in range(sampling_size):
        output_tokens, ins_policy = model.insert_placeholders(prev_decoder_out, encoder_out)
        output_tokens, rep_policy = model.replace_tokens(output_tokens, encoder_out)
        decoder_outs.append(output_tokens)
        policies.append([ins_policy, rep_policy]])
    prev_decoder_out = random.choice(decoder_outs[-sampling_size:])

loss = model.compute_loss(decoder_outs, policies, tgt_tokens)
model.update(loss)
```

Figure 3: The pseudo code of stepwise reward maximization.

B Pseudo code of episodic reward maximization

We show pseudo code of episodic reward maximization in Figure 4.

```
for sample_idx in range(sampling_size):
    prev_decoder_out = model.initialize_output_tokens()
    for step in range(iteration):
        output_tokens, del_policy = model.delete_tokens(prev_decoder_out, encoder_out)
        output_tokens, ins_policy = model.insert_placeholders(output_tokens, encoder_out)
        output_tokens, rep_policy = model.replace_tokens(output_tokens, encoder_out)
        prev_decoder_out = output_tokens

        if step == iteration - 1:
            decoder_outs.append(output_tokens)
        policies.append([del_policy, ins_policy, rep_policy])

loss = model.compute_loss(decoder_outs, policies, tgt_tokens)
model.update(loss)
```

Figure 4: The pseudo code of episodic reward maximization.